

POSIX And Linux 5 System Programming Using C/C++

Using low-level C API for maximum performance, security, extensibility, flexibility, portability

This course examines how to use POSIX standard APIs and Linux libc-specific APIs to create system-level software. POSIX defines specifications (documents) and operating systems (such as Linux with its libc library) or layered libraries (such as [musl](#)) implement these specifications (code). POSIX is supported on a variety of modern operating systems and so system-level application code should strive to use it as much as possible. Implementations can (and almost always do) add extra functions to the API list defined by POSIX and these extra functions provide very useful additional capabilities, though they limit portability.

Developers seeking extra performance and flexible low-level control over OS system calls will benefit from writing their system-level application code in C/C++ (rather than a language that comes with a heavy runtime, such as Java or C#). This course tells them what they need to know to become quickly productive.

We explore the entire path from application code to libc calls, through system calls right to the kernel where the actual functionality is delivered. We look closely at the architecture of how the OS programming interface is exposed to applications.

Contents of One-Day Training Course	
<p>Target Audience System architects and experienced developers who need to create advanced system software using POSIX and Linux libc APIs.</p> <p>Prerequisites Attendees must have some previous experience of system-level programming.</p> <p>Please note: This course does not cover pthreads (multithreading with POSIX) – we offer a separate detailed course for pthreads development.</p>	<p>Big Picture How everything fits together - POSIX, the standard C library, libc, Linux System Calls, the Linux Kernel Alternative implementations: musl, newlib Portability & POSIX on non-linux OSes Role of the Linux Standard Base (LSB) What's needed for development</p> <p>Overview of POSIX Specs POSIX.1-2017 : latest spec with sections on base definitions, system interfaces (APIs), shell & utilities, rationale “Write to the API, not the OS” Relation between POSIX and C (“POSIX is a superset of the standard C library” and libc API is a superset of POSIX) What is specifically outside POSIX scope</p> <p>Linux General Architecture Structure of Linux kernel and userspace Role APIs and system calls play Major OS components and subsystems Layout of OS and apps on file system</p> <p>Linux libc Common datatypes & headers An application developer’s view of the OS Let’s trace a system call from app to API to System Call Interface to kernel Trap 0x80 for x86</p> <p>Major POSIX Functional Areas STREAMS, I/O, signals Regular expressions File descriptors & advanced file handling Process creation (fork/exec) & status</p> <p>Inter-Process Communication IPC overview Message passing semaphores Shared memory</p> <p>What libc Adds To POSIX What’s in libc above and beyond POSIX e.g. cgroups (underpins containers) Multimedia Newer libc system APIs not in POSIX Security concepts</p> <p>Linux Binaries File structure Programmatic interaction Creating binaries Dynamic loading of .so</p> <p>Scheduling Overview of Linux scheduling How apps use APIs to influence scheduling Priority</p> <p>Sockets How to programmatically talk to the net Available APIs and when to use which A protocol-independent API Coding client and server applications Blocking and non-blocking comms Socket options / out-of-band data Addressing, queuing signaling, errors, Advanced sockets</p> <p>Large Codebases Large codebases need more than POSIX Retain portability via e.g. Apache Portable Runtime or ACE</p>